Miquel Esplà Gomis
Bases de Datos (38210)
Máster Universitario en Desarrollo de Aplicaciones y Servicios Web
Departamento de Lenguajes y Sistemas Informáticos
Universitat d'Alacant
Universidad de Alicante

Caché de segundo nivel

En este documento veremos un ejemplo de cómo configurar la caché de segundo nivel en Hibernate. La caché de segundo nivel nos permite almacenar temporalmente datos de nuestra base de datos en memoria o en disco de forma que sean fácilmente recuperables por parte de hibernate para mejorar el rendimiento.

Fichero `pom.xml`

El fichero `pom.xml` contiene las dependencias de Maven. Maven puede descargar automáticamente las dependencias que no se encuentren disponibles localmente de sus repositorios. El fichero `pom.xml` que utilizaremos en este ejemplo es:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>es.ua.eps.cursohibernate</groupId>
  <artifactId>AnotacionVsXML</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>AnotacionVsXML</name>
  <url>http://maven.apache.org</url>

  <properties>
       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
       <!-- SQL Server Connector -->
       <dependency>
       <groupId>net.sourceforge.jtds</groupId>
              <artifactId>jtds</artifactId>
       <version>1.3.1</version>
       </dependency>
       <!-- Hibernate 5.2.10 Final -->
       <dependency>
              <groupId>org.hibernate</groupId>
       <artifactId>hibernate-core</artifactId>
       <version>5.2.10.Final</version>
      </dependency>
```

```
        <!-- EHCache Core APIs -->
        <dependency>
                <groupId>net.sf.ehcache</groupId>
                <artifactId>ehcache-core</artifactId>
                <version>2.6.9</version>
        </dependency>
        <!-- Hibernate EHCache API -->
        <dependency>
                <groupId>org.hibernate</groupId>
                <artifactId>hibernate-ehcache</artifactId>
                <version>5.2.10.Final</version>
        </dependency>

   </dependencies>
</project>
```

Se puede observar que se han añadido dos dependencias relativas a EHCache: el módulo básico de EHCache y el módulo de Hibernate para EHCache.

Entidad *Subject*

Recuperamos el ejemplo de la primera sesión, en la que teníamos una tabla *Subject* para guardar datos de asignaturas.

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Subject](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [name] [varchar](30) NULL,
    [creation_time] [datetime] NULL,
 CONSTRAINT [PK_Subject] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Fichero de Configuración (`hibernate-annotation.cfg.xml`)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
        <session-factory>
        <!-- Datos de conexión con la base de datos- Driver, URL, user, password -->
        <property
name="hibernate.connection.driver_class">net.sourceforge.jtds.jdbc.Driver</property>
        <property
name="hibernate.connection.url">jdbc:jtds:sqlserver://localhost/university</property>
        <property name="hibernate.connection.username">mespla</property>
        <property name="hibernate.connection.password">mespla1</property>
        <!-- Tamaño de la Connection Pool-->
        <property name="hibernate.connection.pool_size">1</property>
```

```
        <!-- Las sesiones con la base de datos se asocian al hilo de ejecución actual -->
        <property name="hibernate.current_session_context_class">thread</property>

        <!-- Opción para depuración: muestra las SQL queries generadas -->
        <property name="hibernate.show_sql">true</property>

        <!-- Dialecto de la base de datos específica: SQL Server -->
        <property name="hibernate.dialect">org.hibernate.dialect.SQLServerDialect</property>

        <!-- Clase que contiene las anotaciones de mapeo -->
        <mapping class="es.ua.eps.cursohibernate.model.SubjectAnnotation"/>

        <!-- activando la caché de segundo nivel -->
        <property name="hibernate.cache.use_second_level_cache">true</property>

        <!-- definiendo EHCache como el gestor de la caché de segundo nivel -->
        <property name="hibernate.cache.region.factory_class">
              org.hibernate.cache.ehcache.EhCacheRegionFactory</property>

        <!-- especificando el fichero de configuración de la caché -->
        <property name="net.sf.ehcache.configurationResourceName">/ehcache.xml</property>

      </session-factory>
</hibernate-configuration>
```

Fichero de configuración de la caché con EHCache (`ehcache.xml`)

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="ehcache.xsd" updateCheck="true"
       monitoring="autodetect" dynamicConfig="true">

       <!--Espacio físico para el almacenaje de la caché-->
       <diskStore path="java.io.tmpdir/ehcache" />

       <!--Configuración por defecto-->
       <defaultCache maxEntriesLocalHeap="10000" eternal="false"
              timeToIdleSeconds="120" timeToLiveSeconds="120" diskSpoolBufferSizeMB="30"
              maxEntriesLocalDisk="10000000" diskExpiryThreadIntervalSeconds="120"
              memoryStoreEvictionPolicy="LRU" statistics="true">
              <persistence strategy="localTempSwap" />
       </defaultCache>

       <!--Configuración específica para los objetos subject-->
       <cache name="subject" maxEntriesLocalHeap="10000" eternal="false"
              timeToIdleSeconds="5" timeToLiveSeconds="10">
              <persistence strategy="localTempSwap" />
       </cache>

       <cache name="org.hibernate.cache.spi.UpdateTimestampsCache"
              maxEntriesLocalHeap="5000" eternal="true">
              <persistence strategy="localTempSwap" />
       </cache>

</ehcache>
```

La configuración define:

- *diskStore*: permite que cuando se alcance el espacio máximo de caché en memoria se puedan guardar datos de la caché en el disco
- *defaultCache*: campo obligatorio, define los parámetros de la región genérica de la caché (donde van a parar las consultas relacionadas con entidades para las cuales no se ha definido una región específica)

- *cache* para *subject*: parámetros de la región específica de la caché para la entidad *subject*
- *UpdateTimestampsCache:* región de la caché donde se guardan las marcas de tiempo (*timestamps*) de todos los elementos de la caché para calcular su caducidad

En las dos regiones definidas en el ejemplo se han definido:

- *maxEntriesLocalHeap*: número de elementos (entities) que se almacenan en memoria. A partir de este número se seguirán almacenando en disco.
- *eternal:* si se define a *true* los elementos no caducan y se ignorarán los *timeouts*
- *timeToIdleSeconds* y *timeToLiveSeconds*: tiempo de caducidad del elemento, bién desde que se ha accedido por última vez (idle) o desde que se ha guardado en la caché (live).
- *maxEntriesLocalDisk*: número máximo de elementos almacenados en disco.
- *diskExpiryThreadIntervalSeconds*: intervalo de comprobación de los tiempos de caducidad (cada cuanto se comprueban)
- *memoryStoreEvictionPolicy*: política de limpieza de la memoria, o qué se va a borrar de la memoria cuando falte espacio. *LRU* corresponde a *Less Recently Used* i establece borrar los elementos menos utilizados
- *persistence strategy*: estrategia de escritura en disco; *localTempSwap* determina que se desplacen algunos elementos de la memoria al disco cuando ésta esté llena.

POJO *Subject*

```
package es.ua.eps.cursohibernate.model;

//Se necesita una serie de librerías para poder definir el mapeo:
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;

import java.util.Date;
//Indicamos que la clase es una entidad a mapear
@Entity
//Indicamos la tabla donde se guardarán los datos del objeto
@Table(name="Subject",
       uniqueConstraints={@UniqueConstraint(columnNames={"id"})})
@Cacheable
@Cache(usage=CacheConcurrencyStrategy.READ_ONLY, region="subject")
public class SubjectAnnotation implements Serializable {

    private static final long serialVersionUID = 1L;

    //Indicamos que el campo id es el identificador y que la base de datos lo genera
automáticamente por auto-incremento
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    //Información sobre la columna de la tabla donde se guardará el ID
    @Column(name="id", nullable=false, unique=true, length=11)
    private int id;

    @Column(name="name", length=30, nullable=true)
    private String name;

    @Column(name="creation_time", nullable=true)
    private Date creationTime;

    public int getId() {
        return id;
```

```
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public Date getCreationTime() {
        return creationTime;
    }
    public void setCreationTime(Date insertTime) {
        this.creationTime = insertTime;
    }
}
```

La entidad se ha asignado a la región de caché *subject* y se ha definido la propiedad *CacheConcurrencyStrategy* a *READ_ONLY*. Esta propiedad puede tomar cuatro valores:

- ***READ_ONLY***: Para casos en que los datos de la entidad no se van a actualizar; no requiere comprobaciones de actualización de la caché y, por tanto, son más rápidos.
- ***READ_WRITE***: Cuando una entidad se actualiza, se bloquea en la caché hasta que la transacción de actualización ha concluido. Cualquier otra operación de lectura será dirigida directamente a la base de datos y no a la caché.
- ***NONSTRICT_READ_WRITE***: Versión laxa del READ_WRITE donde no se controla el acceso inconsistente a la caché en el breve periodo de tiempo en que un valor se está actualizando en la base de datos. Casos de uso muy concretos.
- ***TRANSACTIONAL***: Las actualizaciones de entidades se realizan con transacciones XA distribuidas. Las actualizaciones se realizan al mismo tiempo en la caché y en la base de datos.